Paris School of Business (PSB)

Compréhension et utilisation de 4 packages de R: evir, evd, R.miner, graphics

MSc Data Management

Projet : **R**

par :

Thuy AUFRERE, Nina ZOUMANIGUI, Arnaud Bruel YANKO

Sous la direction de :

M. Henri LAUDE

Enseignant

Année académique 2020-2022

♣ Table des matières ♣

1 Introduction

Dans une session R, nous avons accès à un bon nombre de fonctions et de jeux de données. Les objets accessibles sont ceux contenus dans les packages R chargés à l'ouverture de la session. Un package R est simplement un regroupement de fonctions et de données documentées. Ce tutoriel à pour but de vous faire une présentation de certains packages qui ont été choisis à savoir evir, evd, R.miner, graphics, packages qui nous permettrons :

— D'explorer le monde de la théorie des valeurs extrêmes (Cours intéressant sur la Statisque des valeurs extrêmes) en modélisant les risques extrêmes ;

— Travailler d'une manière générale sur la régression.

2 Installation et chargement d'un package R

L'installation d'un package et le chargement d'un package sont deux étapes distinctes. Certains packages R sont installés automatiquement lors de l'installation de R.

La fonction *installed.packages* retourne des informations à propos des packages R installés sur l'ordinateur local.

Il est simple de charger en R des packages supplémentaires à ceux chargés par défaut. Il suffit d'utiliser les commandes comme dans l'exemple suivant :

```
Install.packages("evir")
library(evir)
```

```
> Arnaud<-installed.packages()
> head(Arnaud, n=3)
Package    LibPath                          Version Priority Depends
Imports
 LinkingTo Suggests    Enhances
abind    "abind"    "C:/Users/yanko/Documents/R/win-library/3.6" "1.4-5"
 NA    "R (>= 1.5.0)"
 "methods, utils"           NA          NA          NA
actuar   "actuar"   "C:/Users/yanko/Documents/R/win-library/3.6" "2.3-3"
NA         "R (>= 3.3.0)"
"stats, graphics, expint" "expint"   "MASS"      NA
askpass "askpass" "C:/Users/yanko/Documents/R/win-library/3.6" "1.1"
NA         NA
   "sys (>= 2.1)"              NA          "testthat" NA
License                License_is_FOSS License_restricts_use OS_type
MD5sum NeedsCompilation Built
abind    "LGPL (>= 2)"         NA                 NA
NA       NA       "no"              "3.6.0"
actuar   "GPL (>= 2)"          NA                 NA
NA       NA       "yes"             "3.6.2"
askpass "MIT + file LICENSE" NA                 NA
NA       NA       "yes"             "3.6.3"
```

2.1 Evir et evd

Evir et Evd : Qu'est ce que c'est ?

Evir est un package de R utiliser dans la théorie de valeurs extrêmes, qui peuvent être divisées dans les groupes suivants ; analyse exploratoire des données, maxima de bloc, pics au-dessus d'un seuil (univarié et bivarié), processus ponctuels, distributions gev/gpd.

Quant-à

Evd il étend les fonctions de simulation, de distribution, de quantile de valeurs extrêmes paramétriques univariées et multivariées, et fournit d'ajustement qui calculent les estimations du maximum de vraisemblance pour des modèles maxima univariées et bivariées, et pour les modèles à seuil univariées et bivariées.

Les fonctions d'Evir et Evd

Ici nous allons recenser certaines fonctions qui peuvent être utilisées dans ces deux packages, commençont par *Evid*.

Listes des fonctions de Evir

Fonctions	Rôles
dgev	Renvoie la distribution des valeurs extrêmes généralisées
dgpd	Distribution de la $Pareto$ généralisée
emplot	Graphique de la fonction de distribution empirique
findthresh	Permet de trouver le seuil
gev	Permet d'ajuster les valeurs des valeurs extrêmes généralisées
gpd	Permet d'ajuste le modèle $Pareto$ généralisé
gumbel	Permet d'ajuste la distribution de $Gumbell$
nidd.annual	Les données de la rivière $Nidd$
pgev	Donne la valeur de la distribution des valeurs extrêmes généralisées
interpret.gpdbiv	Interprétation des résultats de l'ajustement pgd bivarié
rgpd	Distribution de la $Pareto$ généralisée

Listes des fonctions de Evd

Fonctions	Rôles
failure	Temps d'échec
dextreme	Distributions des maxima et es minima
dmvevd	Distributions paramétriques et valeurs extrêmes multivariées
fextreme	Ajustement du maximum de vraisemblance des maxima et des minima
qfrechet	Distribution de $Frchet$
qnweibull	distribution inverse de $weibull$
gumbel	Permet d'ajuste la distribution de $Gumbell$
rorder	Distributions des statistiques d'ordres
confint.evd	Calcule les intervalles de confiance

2.2 Accéder au contenu d'un package R chargé

Une fois un package chargé en R avec la commande *library*, son contenu est accessible dans la session R. Nous avons vu dans des notes précédentes comment fonctionne l'évaluation d'expressions en R. Nous savons donc que le chargement d'un nouveau package ajoute un environnement dans le chemin de recherche de R, juste en dessous de l'environnement de travail. Le chargement de certains packages provoque aussi le chargement de packages dont ils dépendent. Ainsi, parfois plus d'un environnement est ajouté au chemin de recherche de R lors du chargement d'un package. L'environnement d'un package contient les fonctions publiques et les données du package.

Jeux de données

Souvent, les jeux de données inclus dans un package se retrouvent directement dans l'environnement d'un package dans le chemin de recherche. C'est le cas, par exemple, des jeux de données du package datasets.

head(ls("package :datasets"), n = 8)

```
> head(ls("package:datasets"), n=8)
[1] "ability.cov"   "airmiles"      "AirPassengers" "airquality"
"anscombe"      "attenu"        "attitude"      "austres"
```

Dans notre cas espèce seulement les 8 premiers éléments de la liste sont affichés ici, car cette liste compte normalement 104 éléments.
Cependant, les jeux de données sont parfois cachés. Ils sont alors traités différemment des fonctions privées et ne se retrouvent même pas dans l'espace de noms du package.
La fonction data est très utile dans ce cas. Cette fonction a en fait plusieurs utilités.
Premièrement, elle permet d'énumérer tous les jeux de données contenus dans un package.
data(package = "evir")
Dans notre cas espèce, nous allons à partir du packages "evir", afficher le jeu de données $nidd.annual$, ces données représentent les niveaux maximums annuels de la *rivière Nidd* dans le Yorkshire..
On a :

```
library(evd)
library(evir)
library(actuar)
library(rmutil)
data(nidd.annual)
nidd<-nidd.annual
```

Et on a la sortie suivante :

```
> nidd
 [1]  65.08  65.60  75.06  76.22  78.55  81.27  86.93  87.76  88.89
90.28  91.80  91.80  92.82  95.47 100.40 111.54 111.74 115.52 131.82
138.72 148.63 149.30 151.79 153.04 158.01 162.99 172.92 179.12 181.59
189.04 213.70 226.48 251.96 261.82 305.75
```

Nous obtenons en sortie un vecteur numérique contenant 35 observations.

2.3 Applications

Pour application on utilise les données des crues annuelles de la rivière $Nidd$, dans le Yorkshire.

Contexte

La hauteur d'une rivière est modélisée par une variable aléatoire X.
On dispose de $\{X_1, \cdots, X_n\}$ un échantillon de hauteurs d'eau annuelles. On note $X_{1,n} \leq X_{2,n} \leq, \cdots, \leq X_{n,n}$ l'échantillon ordonné, avec $n \in \mathbb{N}$.

Deux problèmes complémentaires :

* Calculer la probabilité p d'une hauteur d'eau h extrême
  $p = \mathbf{P}(X \geq h)$ avec $h > X_{n,n}$.
* Calculer le niveau d'eau h qui est atteint ou dépassé une seule fois sur $T > n$, i.e. résoudre
  $\frac{1}{T} = \mathbf{P}(X \geq h)$

Modélisation : Le but ici étant d'écrire un programme permettant de calculer la fonction empirique des excès moyens et de tracer les différents seuils :

Application 1

(−) On Charge au préalable nos deux packages, les bouts de code suivant permettent de faire ce travail :

```
library(evd)
library(evir)
```

(−) Nous allons commencer par lire les données du fichier $Nidd$ et les classer par ordre décroissant dans un vecteur noté $nidd$. On crée un vecteur $seuil$ ordonné par ordre.

```
nidd2<-sort(nidd, decreasing = TRUE)
nidd2
seuil=seq(70,300,by =5)
#seuil<-70:300
seuil
taille_seuil=length(seuil)
taille_nidd2=length(nidd2)
```

(−) Afin de calculer la moyenne des excès pour chaque seuil, on crée une matrice dont le nombre de lignes correspond à la taille du vecteur "$seuil$" et le nombre de colonnes correspond à la taille du vecteur "$nidd$" telle que chaque ligne i correspond aux excès au

delà du $seuil[i]$.

Puisque certains excès peuvent être négatifs, nous mettons les zéros à la place des termes négatifs :

```
X=matrix(0,nrow=length(seuil),ncol=length(nidd2))
X
#View(X)

for(i in 1:length(seuil))
{for(j in 1:length(nidd2))
        {X[i,j]=max(nidd2[j]-seuil[i],0)}
}
X
```

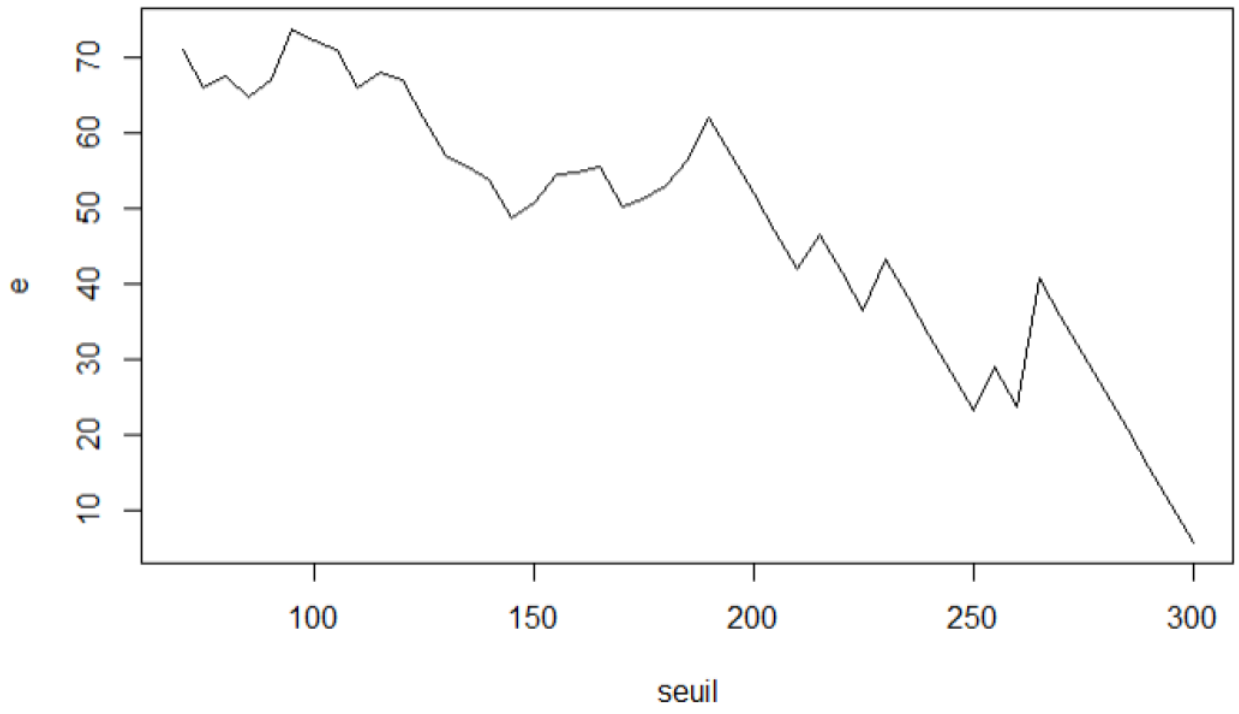
(–) On calcule pour chaque seuil la moyenne des excès :

```
somme=rep(0,length(seuil))
somme
Compteur=rep(0,length(seuil))
Compteur
e=c()
e
for(i in 1:length(seuil))
{
        for(j in 1:length(nidd2))
        {
                if ( X[i,j]>0 )
                {
                        somme[i]=somme[i]+X[i,j]
                        Compteur[i]=Compteur[i]+1
                }
        }
        e[i]=somme[i]/Compteur[i]
}
e
```

(−) La fonction moyenne des excès est une méthode permettant de répondre à l'une des difficultés de la modélisation des sinistres extrêmes qui est la détermination du seuil. Ainsi, graphiquement il est possible de déterminer le seuil le plus adéquat, en prenant la valeur à partir de laquelle la fonction moyenne des excès est linéaire.

Regardons ensemble ce que cela produit :

```
plot(seuil,e,type='l')
```



Dans notre cas, La fonction moyenne des excès est linéaire à partir de la valeur 260.

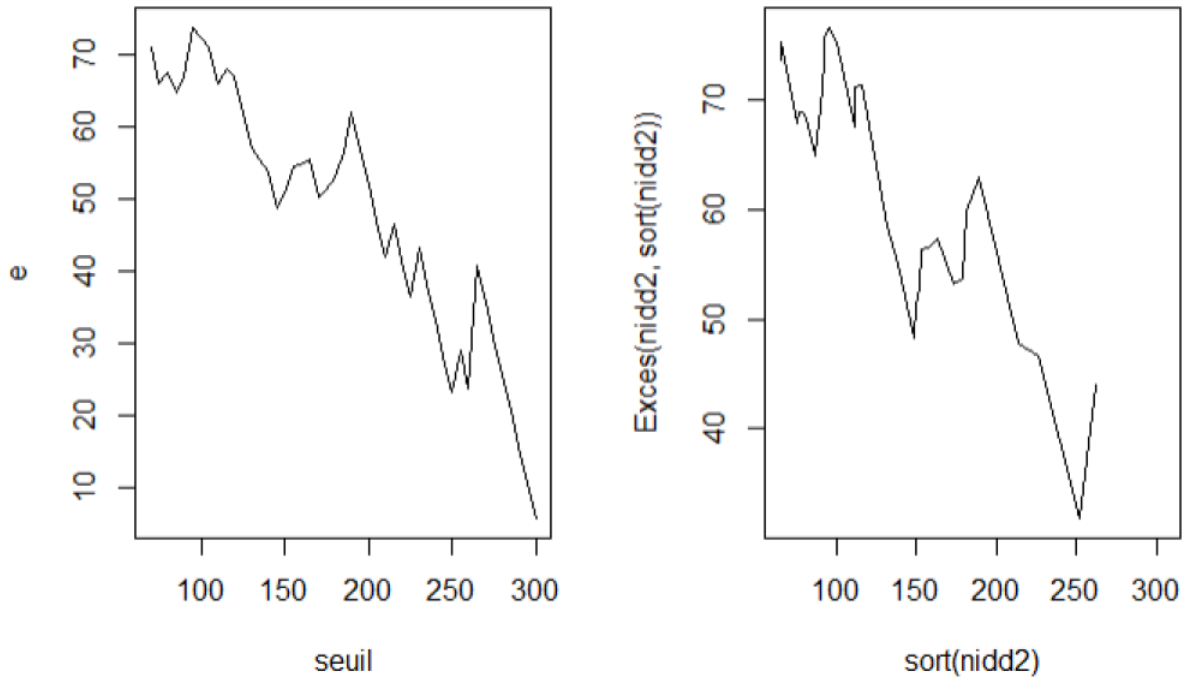
(−) Ici on trace sur deux graphiques juxtaposés : la fonction empirique des excès moyens en fonction des statistiques d'ordre $x_{(n-k)}$ et en fonction de k.

Le $1er$ représente la moyenne des dépassements des crues annuelles de la rivière $Nidd$ par rapport à un certain seuil (indépendant de nos données de départ). Le $2eme$ représente la moyenne des dépassements de la crue annuelle de la rivière $Nidd$ d'une année par rapport aux autres.

```
Exces=function(observations,seuil)
{
        Exces2=matrix(0,nrow=length(seuil),
        ncol=length(observations))
        Exces2
        #View(X)

        for(i in 1:length(seuil))
        {
                for(j in 1:length(observations))
                {
                        Exces2[i,j]=max(observations[j]-seuil[i],0)
                }
        }
        Exces2
        somme2=rep(0,length(seuil))
        somme2
        Compteur2=rep(0,length(seuil))
        Compteur2
        e2=c()
        e2
        for(i in 1:length(seuil))
        {
                for(j in 1:length(observations))
                {
                        if ( Exces2[i,j]>0 )
                        {
                                somme2[i]=somme2[i]+Exces2[i,j]
                                Compteur2[i]=Compteur2[i]+1
                        }
                }
                e2[i]=somme2[i]/Compteur2[i]
```

```
        }
        e2
}
```

```
Exces(nidd2, sort(nidd2))
par(mfrow=c(1,2))
plot(seuil,e,type='l')
plot(sort(nidd2),Exces(nidd2, sort(nidd2)),type='l')
```



Les deux graphique sont la même allure, et la fonction moyenne des excès devient linéaire aux environs de la valeur 260.

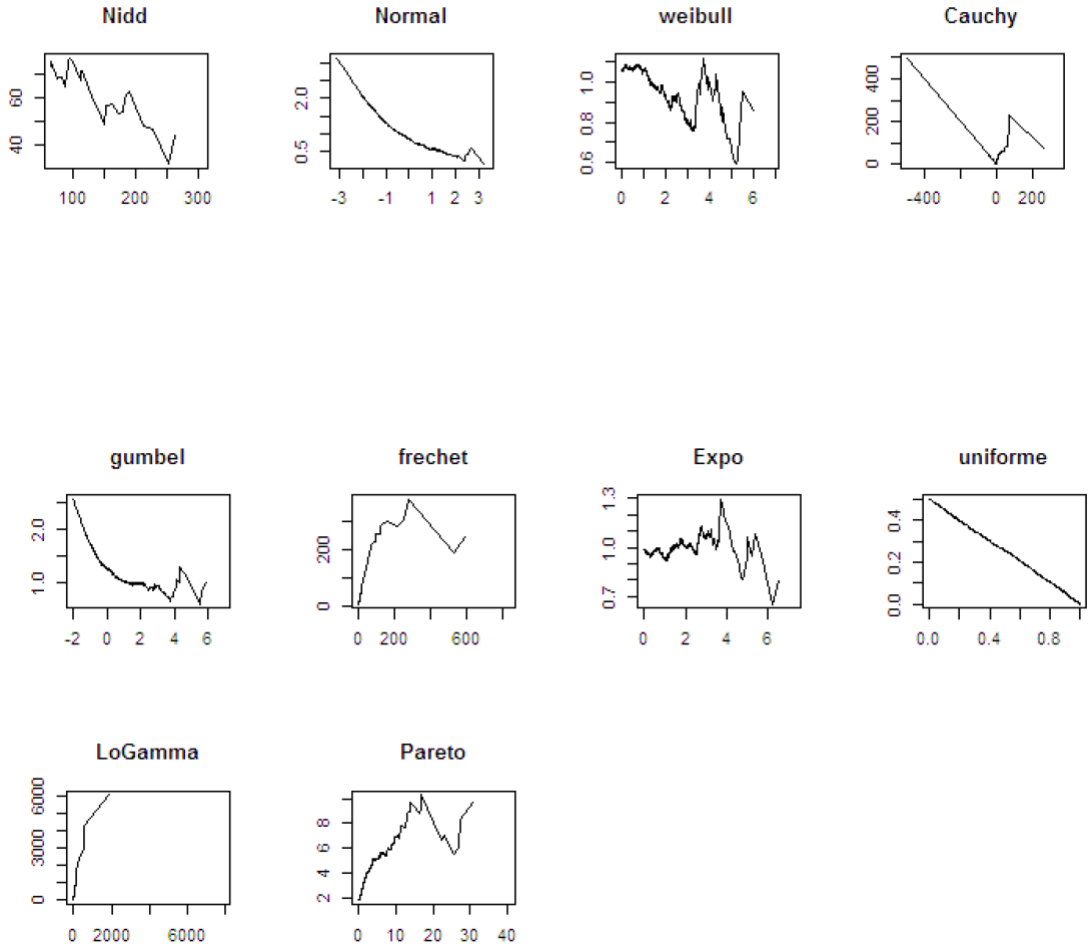
Pour aller plus loin l'idéal serait d'appliquer cela aux différentes lois usuelles : La loi Pareto, Weibull, Frechet, Gamma, Cauchy, Normal et bien d'autres : (Les differents lois en Statisque des valeurs extrêmes)

(–) Application aux lois usuelles

```
###########loi Normal Centrée réduite###########
Nidd_normal=rnorm(1000,0,1)
Nidd_normal_dec=sort(Nidd_normal, decreasing = TRUE)
seuil_stat_ord=sort(Nidd_normal, decreasing = FALSE)
#Exces(Nidd_normal_dec,seuil_stat_ord)
par(mfrow=c(3,4))
plot(sort(nidd2),Exces(nidd2,sort(nidd2)),type='l',main='Nidd',
xlab = "" , ylab = "")
plot(seuil_stat_ord,Exces(Nidd_normal_dec,seuil_stat_ord),type='l',
main='Normal',xlab = "" , ylab = "")

###########loi Weibull###########
Nidd_weibull=rweibull(1000, 1)
Nidd_weibull_dec=sort(Nidd_weibull, decreasing = TRUE)
seuil_stat_ord=sort(Nidd_weibull, decreasing = FALSE)
#Exces(Nidd_weibull_dec,seuil_stat_ord)
#par(mfrow=c(2,2))
#plot(sort(nidd2),Exces(nidd2,sort(nidd2)),type='l')
plot(seuil_stat_ord,Exces(Nidd_weibull_dec,seuil_stat_ord),
type='l',main='weibull',xlab = "" , ylab = "")

###########loi Cauchy###########
Nidd_cauchy=rcauchy(1000, 0, 1)
Nidd_cauchy_dec=sort(Nidd_cauchy, decreasing = TRUE)
seuil_stat_ord=sort(Nidd_cauchy, decreasing = FALSE)
#Exces(Nidd_cauchy_dec,seuil_stat_ord)
#par(mfrow=c(2,2))
#plot(sort(nidd2),Exces(nidd2,sort(nidd2)),type='l')
plot(seuil_stat_ord,Exces(Nidd_cauchy_dec,seuil_stat_ord),
type='l',main='Cauchy',xlab = "" , ylab = "")
```

```
###########loi Gumbel###########
Nidd_gumbel=rgumbel(1000,0,1)
Nidd_gumbel_dec=sort(Nidd_gumbel, decreasing = TRUE)
seuil_stat_ord=sort(Nidd_gumbel, decreasing = FALSE)
#Exces(Nidd_gumbel_dec,seuil_stat_ord)
#par(mfrow=c(2,2))
#plot(sort(nidd2),Exces(nidd2,sort(nidd2)),type='l')
plot(seuil_stat_ord,Exces(Nidd_gumbel_dec,seuil_stat_ord),
type='l',main='gumbel',xlab = "" , ylab = "")

###########loi Frechet###########
Nidd_frechet=rfrechet(1000, loc=0, scale=1, shape=1)
Nidd_frechet_dec=sort(Nidd_frechet, decreasing = TRUE)
seuil_stat_ord=sort(Nidd_frechet, decreasing = FALSE)
#Exces(Nidd_frechet_dec,seuil_stat_ord)
#par(mfrow=c(2,2))
#plot(sort(nidd2),Exces(nidd2,sort(nidd2)),type='l')
plot(seuil_stat_ord,Exces(Nidd_frechet_dec,seuil_stat_ord),
type='l',main='frechet',xlab = "" , ylab = "")

###########loi Exponentielle###########
Nidd_Expo=rexp(1000,1)
Nidd_Expo_dec=sort(Nidd_Expo, decreasing = TRUE)
seuil_stat_ord=sort(Nidd_Expo, decreasing = FALSE)
#Exces(Nidd_Expo_dec,seuil_stat_ord)
#par(mfrow=c(2,2))
#plot(sort(nidd2),Exces(nidd2,sort(nidd2)),type='l')
plot(seuil_stat_ord,Exces(Nidd_Expo_dec,seuil_stat_ord),
type='l',main='Expo',xlab = "" , ylab = "")
```

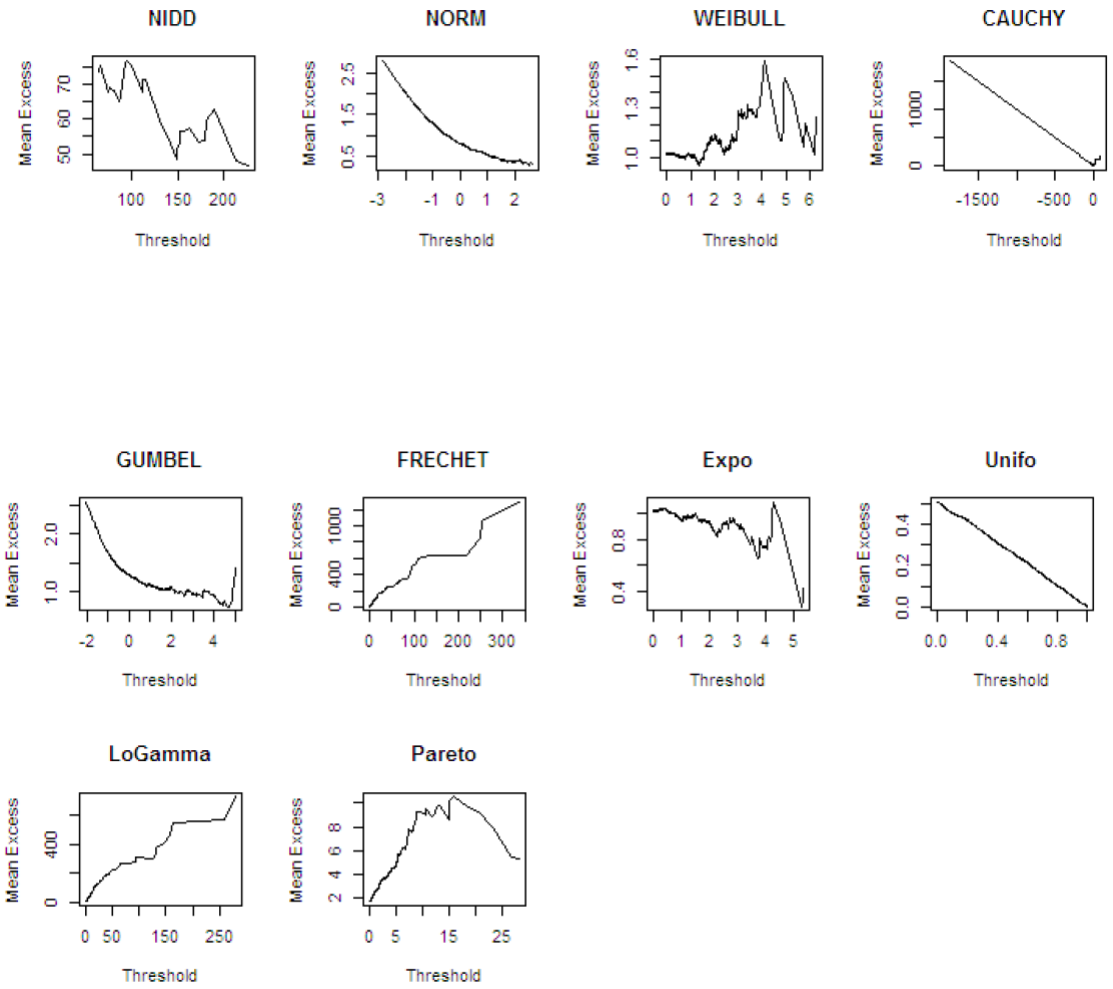
```
###########loi Uniforme###########
Nidd_unifo=runif(1000,0,1)
Nidd_unifo_dec=sort(Nidd_unifo, decreasing = TRUE)
seuil_stat_ord=sort(Nidd_unifo, decreasing = FALSE)
#Exces(Nidd_unifo_dec,seuil_stat_ord)
#par(mfrow=c(2,2))
#plot(sort(nidd2),Exces(nidd2,sort(nidd2)),type='l')
plot(seuil_stat_ord,Exces(Nidd_unifo_dec,seuil_stat_ord),
type='l',main='uniforme',xlab = "" , ylab = "")

###########loi Gamma###########
Nidd_LoGamma=rlgamma(1000,1,1)
Nidd_LoGamma_dec=sort(Nidd_LoGamma, decreasing = TRUE)
seuil_stat_ord=sort(Nidd_LoGamma, decreasing = FALSE)
#Exces(Nidd_Gamma_dec,seuil_stat_ord)
#par(mfrow=c(2,2))
#plot(sort(nidd2),Exces(nidd2,sort(nidd2)),type='l')
plot(seuil_stat_ord,Exces(Nidd_LoGamma_dec,seuil_stat_ord),
type='l',main='LoGamma',xlab = "" , ylab = "")

###########loi Pareto###########
Nidd_Pareto=rpareto(1000,2,2)
Nidd_Pareto_dec=sort(Nidd_Pareto, decreasing = TRUE)
seuil_stat_ord=sort(Nidd_Pareto, decreasing = FALSE)
#Exces(Nidd_Pareto_dec,seuil_stat_ord)
#par(mfrow=c(2,2))
#plot(sort(nidd2),Exces(nidd2,sort(nidd2)),type='l')
plot(seuil_stat_ord,Exces(Nidd_Pareto_dec,seuil_stat_ord),
type='l',main='Pareto',xlab = "" , ylab = "")
```



(−) utilisons la fonction "*meplot*" du package "*evir*" pour tracer la fonction empirique des excès moyens pour différentes lois.

```
Normal = rnorm(1000,0,1)
Weibull = rweibull(1000, 1)
Cauchy = rcauchy(1000, 0, 1)
Gumbel = rgumbel(1000,0,1)
Frechet = rfrechet(1000, loc=0, scale=1, shape=1)
Expo=rexp(1000,1)
Unifo=runif(1000,0,1)
LoGamma=rlgamma(1000,1,1)
Pareto=rpareto(1000,2,2)
```

```
close.screen( all = TRUE )
par(mfrow=c(3,4))
meplot(nidd, 3, type="l", main="NIDD")
meplot(Normal, 3, type="l", main="NORM")
meplot(Weibull, 3, type="l", main="WEIBULL")
meplot(Cauchy, 3, type="l", main="CAUCHY")
meplot(Gumbel, 3, type="l", main="GUMBEL")
meplot(Frechet, 3, type="l", main="FRECHET")
meplot(Expo, 3, type="l", main="Expo")
meplot(Unifo, 3, type="l", main="Unifo")
meplot(LoGamma, 3, type="l", main="LoGamma")
meplot(Pareto, 3, type="l", main="Pareto")
```



Application 2 : On termine le seuil avec le Graphique de $HILL$

Le graphique de $Hill$ est un estimateur qui est défini uniquement pour $\psi > 0$, c'est-à-dire dans le cas où la distribution des valeurs extrêmes correspond à une distribution de Fréchet. Le seuil, dans ce cas, correspond à la valeur à partir de laquelle la fonction de $Hill$ est assimilable à une droite horizontale.

Implémentation

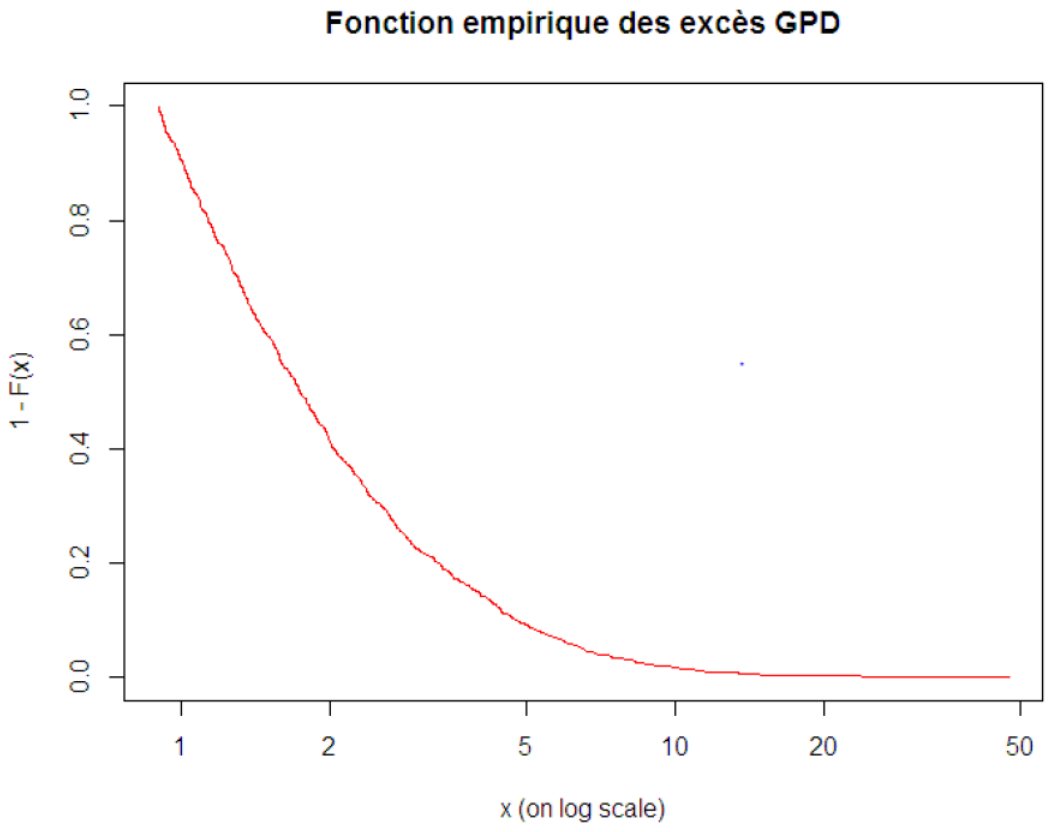
1. Commençons par simuler une GPD à l'aide de $rgpd$:

```
GP=rgpd(1000, 0.4, 0.9)
#GP=rgpd(1000, 0.30)
#GP=rgpd(500,0.4,0,1)
GP
```

2. On trace la fonction de survie empirique des excès à l'aide de la fonction $emplot$:

```
emplot(GP,type="l",main="Fonction empirique des excès GPD",
col="red")
```

Le graphe obtenu est le suivant :



3. On utilise la fonction *gpd* pour calculer les estimateurs de γ et σ :

```
Estim_ml=gpd(GP,0,method = "ml")$par.ests
Estim_pwm=gpd(GP,0,method = "pwm")$par.ests
Estim_ml
Estim_pwm
```

Par la méthode de maximum de vraisemblance

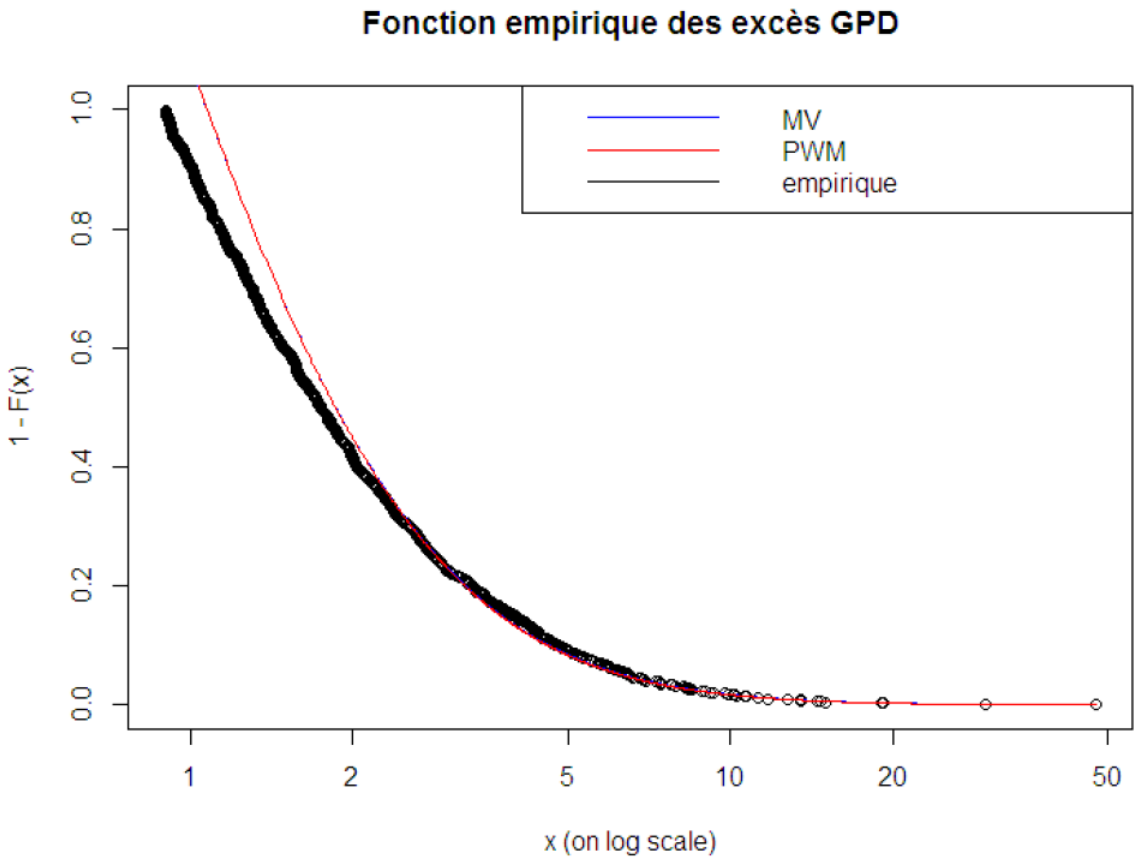
```
> Estim_ml
xi          beta
0.03114114  2.39842599
> Estim_pwm
xi          beta
-0.5266072   3.7816527
```

4. On trace les fonctions de survie des deux GPD estimés sur un même graphe :

```
fgpd=function(loi,x){
        n=length(loi)

        sigma_ml=gpd(loi, nextremes=n, method="ml")$par.ests[1]
        gamma_ml=gpd(loi, nextremes=n, method="ml")$par.ests[2]
        sigma_pwm=gpd(loi, nextremes=n, method="pwm")$par.ests[1]
        gamma_pwm=gpd(loi, nextremes=n, method="pwm")$par.ests[2]

        emplot(GP, main="Fonction empirique des excès GPD")
        lines(x,1-pgpd(x,sigma_ml,gamma_ml),col="blue",
        main="Fonction
        de survie",type="l")
        lines(x,1-pgpd(x,sigma_pwm,gamma_pwm),col="red")
        legend(legend=c("MV","PWM","empirique"),"topright",
        col=c("blue","red","black"),lty=c(1,1,1))
}
x=sort(GP)
```

```
x=seq(x[1],x[length(x)], 0.01)
fgpd(GP,x)
```

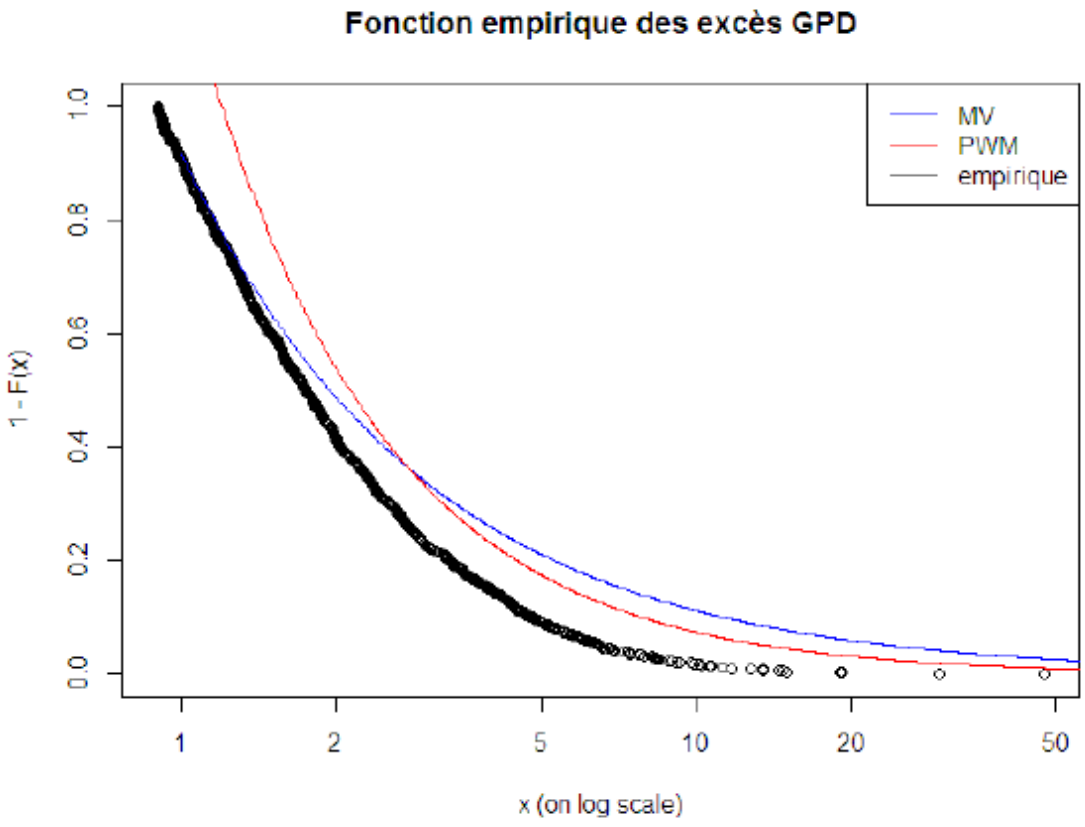
Visualisation



5. Appliquons ceci à d'autres lois usuelles

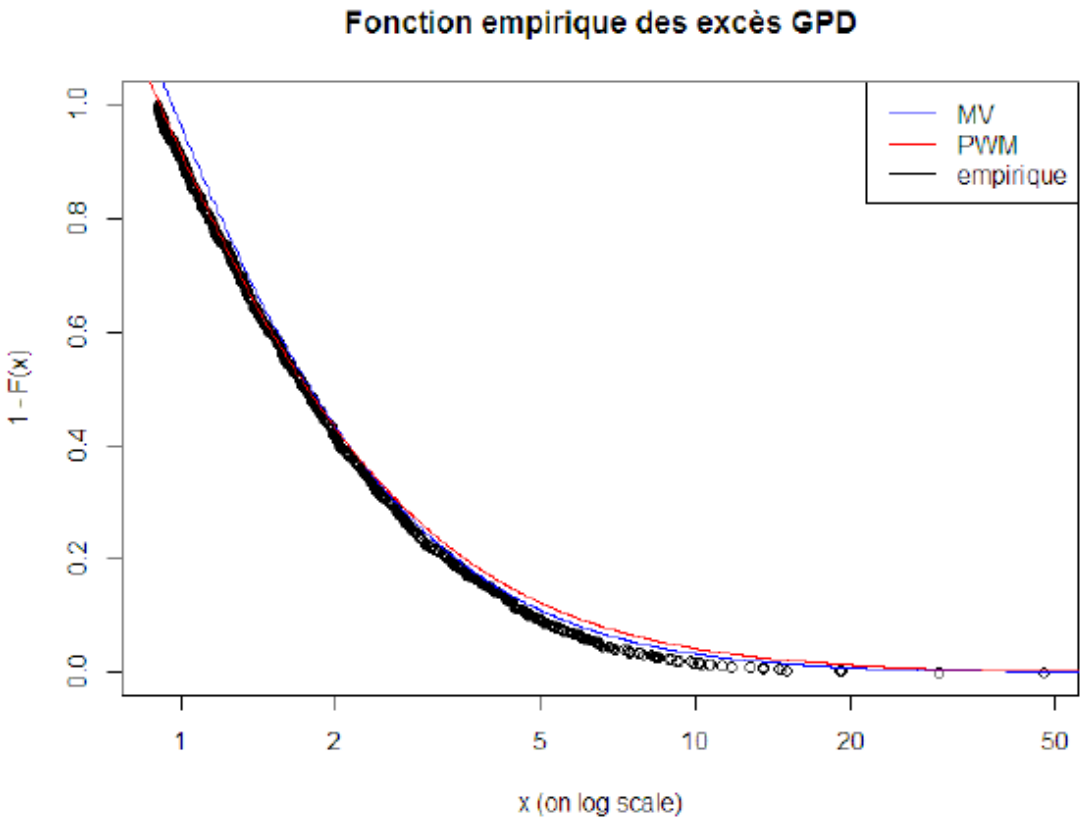
Loi LogGamma :

```
logamma=rlgamma(1000,1,1)
x=sort(logamma)
x=seq(x[1],x[length(x)], 0.01)
fgpd(logamma,x)
```



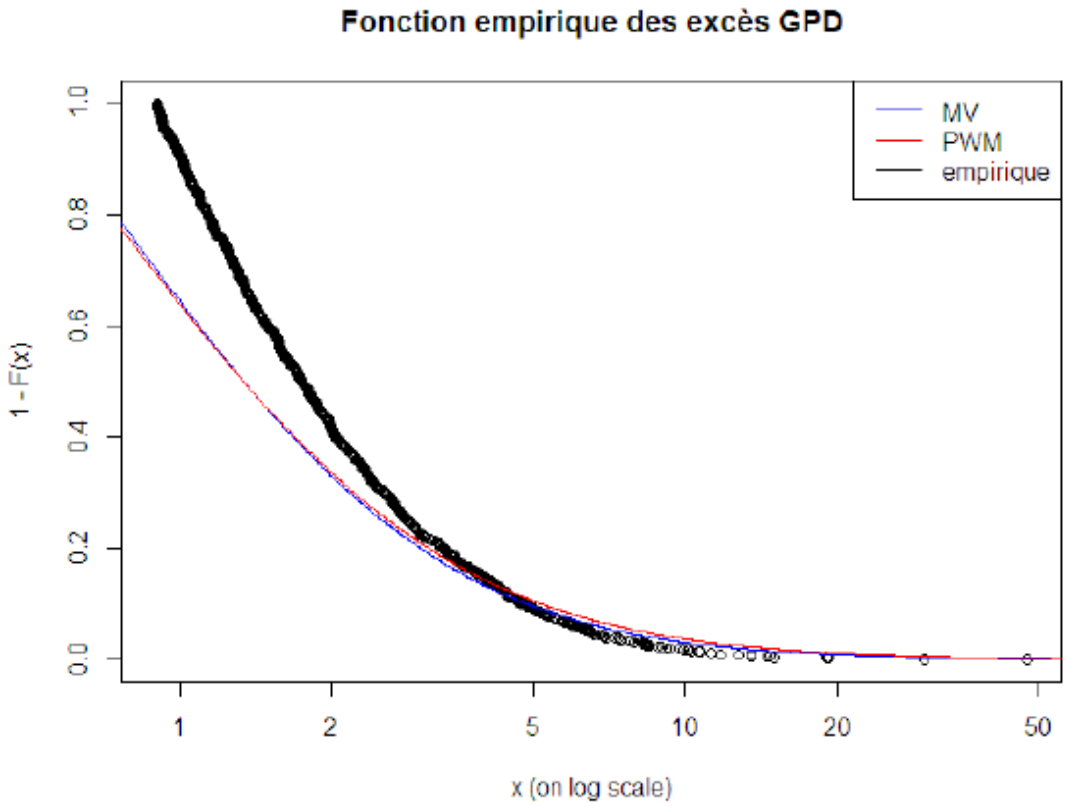
Loi Pareto :

```
Pareto=rpareto(1000,2,2)
x=sort(Pareto)
x=seq(x[1],x[length(x)], 0.01)
fgpd(Pareto,x)
```



Loi Burr :

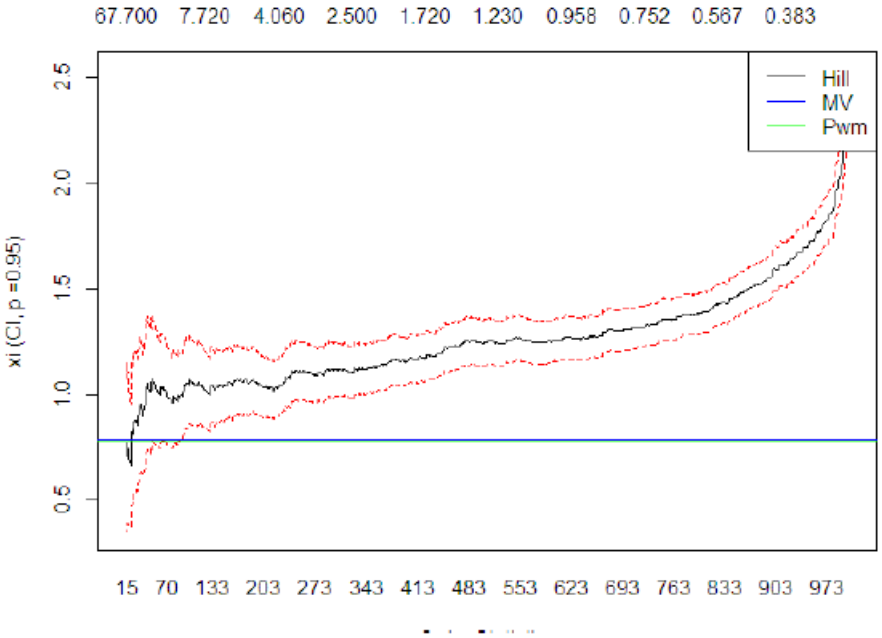
```
bur=rburr(1000,1,1,2)
x=sort(bur)
x=seq(x[1],x[length(x)], 0.01)
fgpd(bur,x)
```



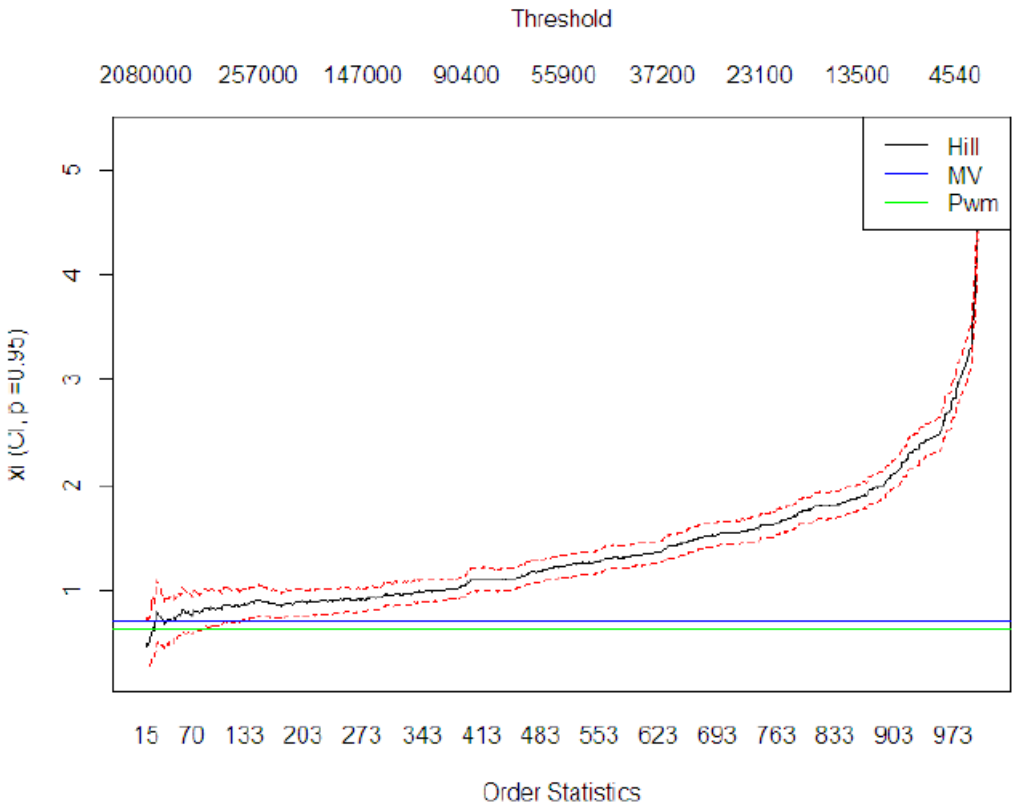
6. On trace sur un même graphique les 3 estimateurs γ : PWM, EMV , $Hill$

```
f_hill=function(loi,n){
        ml=gpd(loi, nextremes=n, method="ml")$par.ests[1]
        pwm=gpd(loi, nextremes=n, method="pwm")$par.ests[1]
        #plot(ml)
        hill(loi, "xi", xlim=c(15,n))
        abline(h=ml, col="blue")
        abline(h=pwm, col="green")
        #dev.off()
        legend(legend=c("Hill", "MV", "Pwm"),"topright",
        col=c("black","blue","green"), lty=c(1,1,1))
        #legend(legend=c("MV","PWM","empirique"),
        "topright",col=c("red","blue","black"),lty=c(1,1,1))

}
#######Loi de Frechet #######
frechet=rfrechet(1000, loc=0, scale=1, shape=1)
f_hill(frechet,1000)

####### Loi Log-gamma #######
loi=rlgamma(1000,55,5)
f_hill(loi,1000)

#######Loi Pareto #######
Pareto=rpareto(1000,10,15)
f_hill(Pareto,1000)

####### loi de Burr #######
bur=rburr(1000,1,1,2)
f_hill(bur,1000)

f_hill(nidd,length(nidd))
```

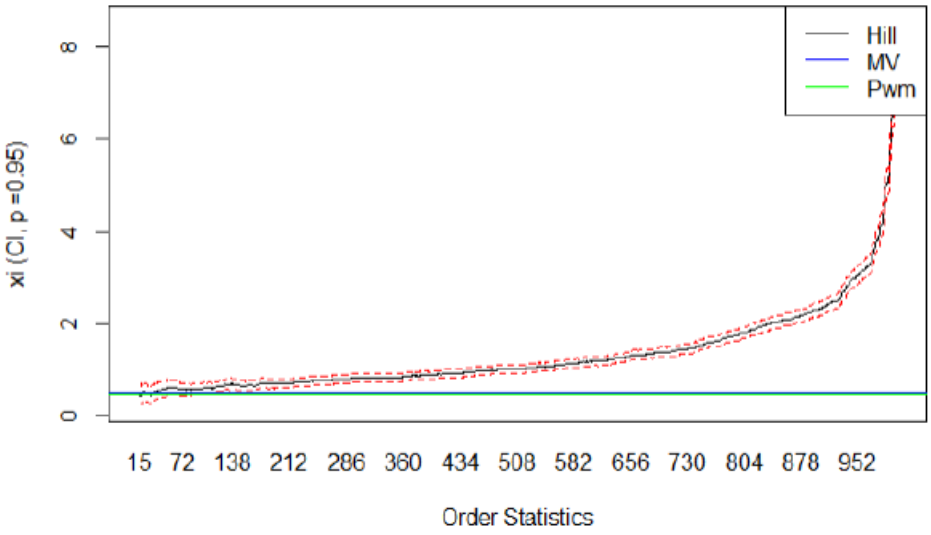
Loi Frechet :



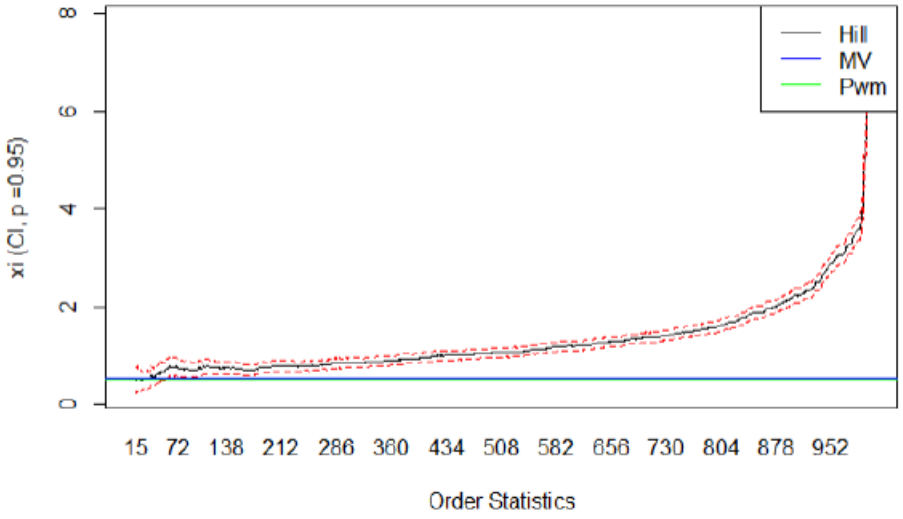
Loi Log-gamma



Loi Pareto



Loi Burr



En conclusion, à partir des graphes obtenus précédemment, pour les lois considérées,la fonction $Hill$ devient une droite horizontale aux environs de la valeur 260, est l'estimateur de $Hill$ est proche de EMV et PWM. On peut donc considérer que le domaine d'attraction de la distribution des crues est $Frechet$.

3 Package RMiner : Data Mining Classification et Méthode de Regression

Rminer : kesako ?

Le package Rminer aide à la manipulation des données algorithmiques pour le data mining concernant la classification et la régression (incluant la prévision de series temporelles). Pour utiliser le package Rminer et ses fonctions :

- vérifier si le package est installé dans R ;
- sinon installer le package puis l'appeler avec la library

```
Install.packages(« rminer »)
Library(rminer)
```

Les fonctions de Rminer

Ci-desous la liste des fonctions du package Rminer. Nous n'allons vous présenter que les fonctions qui nous semblent les plus pertinentes dans ce tutoriat.

Si vous souhaitez approfondir votre connaissance sur les autres fonctions, nous vous invitons à aller sur la documentation R du package : Lien

Si vous souhaitez consulter l'aide, taper dans la commande R `(help(package=rminer))` .

Liste des fonctions de Rminer

CaseSeries
crossvaldata
delevels
holdout
Importance
imputation
lforecast
mgraph
mining
mmetric
mparheuristic
predict.fit
savemining
safri1
sin1reg
vecplot

Ce tutoriat se concentra sur les fonctions suivantes du package Rminer qui peuvent être réparties dans 3 phases dans l'analyse des données :

- Data preparation : CasesSeries, delevels, imputation,
- Modeling : fit, predict, mining,
- Evaluation : mmetric, mgraph, mining

3.1 Data preparation : CasesSeries, delevels

CasesSeries : créer un data frame d'une série temporelle en utilisant la fenêtre glissante. Fenêtre glissante : (« sliding window ») : Fenêtre temporelle utilisée dans l'exploration des flots de données (« Data stream mining ») pour en extraire des motifs. Les fenêtres peuvent avoir une taille W fixe et la fouille de données s'effectue sur les W dernières transactions, ou sur les

W dernières unités de temps, elles peuvent aussi avoir une taille variable, dans la détection de la dérive conceptuelle.

```
# Exemple
> t=1:20
> d=CasesSeries(1:10,c(1,3,4))
> print(d)
[1] lag4 lag3 lag1  y
1    1    2    4  5
2    2    3    5  6
3    3    4    6  7
4    4    5    7  8
5    5    6    8  9
6    6    7    9 10

> d=CasesSeries(1:10,c(1,2,3))
> print(d)
[1] lag3 lag2 lag1  y
1    1    2    3  4
2    2    3    4  5
3    3    4    5  6
4    4    5    6  7
5    5    6    7  8
6    6    7    8  9
7    7    8    9 10
```

==> la fonction retourne un data frame où y est cible de l'output et les inputs sont le temps de latence

Delevels : réduire, remplace ou transforme les niveaux du data frame et la variable du facteur.

```
> f=factor(c("A","A","B","B","C","D","E"))
> print(table(f))
[1] f
```

```
A B C D E
2 2 1 1 1

# remplacer "A" en "a":
> f1=delevels(f,"A","a")
> print(table(f1))
[1] f1
a B C D E
2 2 1 1 1

# combiner c("C","D","E") en "CDE":
> f2=delevels(f,c("C","D","E"),"CDE")
> print(table(f2))
[1] f2
A   B CDE
2   2   3

# combiner c("B","C","D","E") en _OTHER:
> f3=delevels(f,c("B","C","D","E"))
> print(table(f3))
[1] f3
A _OTHER
2      5
```

3.2 Modeling : fit, predict

Rminer comprend 14 méthodes de classification et 15 méthodes de régression, tout directement disponible à travers les facteurs *fit, predict, mining* .

Fit : la fonction ajuste un modèle sélectionné de jeux de données et peut automatiquement ajuster les hyperparamètres. Les hyperparamètres sont des paramètres réglables qui vous permettent de contrôler le processus d'entraînement du modèle.

Predict : la fonction donne un modèle ajusté et calcule les prédictions pour un jeu de données. Les fonctions *fit* et *predict* proposent plusieurs modèles tels que :

- rpart (arbre de décision)

- randomForest
- lm (régression linéaire ou multiple)
- cv.glmnet : modèle linéaire généralisé.

Par defaut, le type du modèle Rminer (classification ou régression) dépend du type d'output.

- Si c'est un facteur (discret) alors c'est une probabilité de classificaiton
- Si c'est numérique (int, num) alors c'est une régression qui est executée.

Voir tous les modèles de *fit* et *predict* : lien fit et lien predict

```
# Exemple qui montre comment la transformation fonctionne avec 'fit'
  et 'predict':
M = fit(y~.,data=sa_ssin,model="mr") # régression linéaire
P = predict(M,data.frame(x1=-1000,x2=0,x3=0,x4=0,y=NA)) # P devrait
être négatif
print(P)
[1] -0.4144042
M = fit(y~.,data=sa_ssin,model="mr",transform="positive")
P = predict(M,data.frame(x1=-1000,x2=0,x3=0,x4=0,y=NA)) # P n'est
pas négatif
print(P)
[1] 0
```

3.3 Evaluation : mmetric, mgraph, mining

Rminer comprend une large sélection de métriques d'évaluation et de graphes qui peut être utilisée pour évaluer la qualité des modèles ajsutés et extraire les données apprises du modèle data-driven.

Mmetric : fonction qui calcule les erreurs métriques de classification ou régression. Ci-dessous les quelques mesures de mmetric :

- ALL : sort toutes les mesures de *mmetric*
- F1 score, $[0 - 100\%]$

- TPR (true positive rate) : c'est la sensibilité, correspondant au taux de vrais positifs, $[0 - 100\%]$
- PRECISION : la précision, c'est-à-dire le nombre de documents pertinents retrouvés rapporté au nombre de documents total proposé pour une requête donnée
- ACC : taux d'exactitude de classification, $[0 - 100\%]$
- ACCLASS : taux d'exactitude de classification par classe, $[0 - 100\%]$
- TPR (true positive rate) : c'est la sensibilité, correspondant au taux de vrais positifs, $[0 - 100\%]$.

Voir tous les modèles de *mmetric* : lien

```
# Exemple
> y=factor(c("a","a","a","a","b","b","b","b"))
> x=factor(c("a","a","b","a","b","a","b","a"))
> print(mmetric(y,x,"CONF")$conf)
[1]  pred
target a b
a 3 1
b 2 2
> print(mmetric(y,x,metric=c("ACC","TPR","ACCLASS")))
[1] ACC      TPR1     TPR2 ACCLASS1 ACCLASS2
62.5     75.0     50.0     62.5     62.5

> print(mmetric(y,x,"ALL"))
[1]      ACC       CE      BER    KAPPA  CRAMERV  ACCLASS1
ACCLASS2  BAL_ACC1
62.5000000 37.5000000 37.5000000 25.0000000  0.0000000 62.5000000
62.5000000 62.5000000
BAL_ACC2     TPR1     TPR2     TNR1     TNR2 PRECISION1
PRECISION2       F11
62.5000000 75.0000000 50.0000000 50.0000000 75.0000000 60.0000000
66.6666667 66.6666667
F12      MCC1      MCC2
57.1428571  0.5163978  0.5163978
```

Sin1reg : un simple jeu de données avec 1000 points où $y = 0.7 * sin(pi * x1/2000) + 0.3 * sin(pi * x2/2000)$

Mgraph : fonction graphique

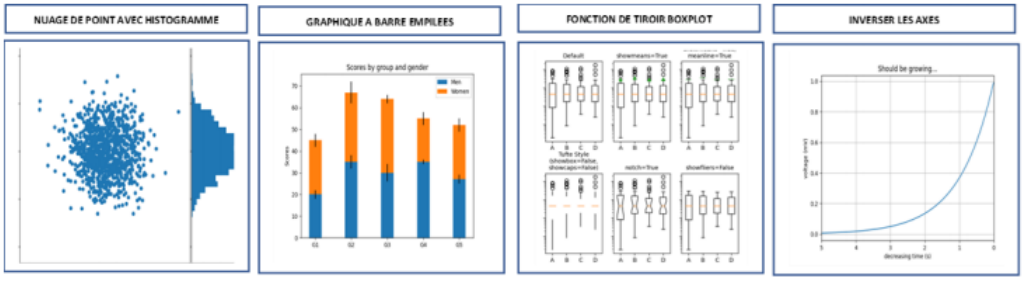
Mining : la fonction réalise plusieurs exécutions ajustées et de prédictions, selon une méthode de validation et donne un nombre d'exécutions.

```
# Exemple de régression avec les fonctions sin1reg, mgraph et mining
data(sin1reg)
M1=mining(y~.,sin1reg[,c(1,2,4)],model="mr",Runs=5)
M2=mining(y~.,sin1reg[,c(1,2,4)],model="mlpe",nr=3,maxit=50,size=4,
Runs=5,feature="simp")
L=vector("list",2); L[[1]]=M2; L[[2]]=M1
mgraph(L,graph="REC",xval=0.1,leg=c("mlpe","mr"),main="REC curve")
mgraph(L,graph="DLC",metric="TOLERANCE",xval=0.01,
leg=c("mlpe","mr"),main="DLC: TOLERANCE plot")
mgraph(M2,graph="IMP",xval=0.01,leg=c("x1","x2"),
main="sin1reg Input importance",axis=1)
mgraph(M2,graph="VEC",xval=1,main="sin1reg 1-D VEC curve for x1")
mgraph(M2,graph="VEC",xval=1,
main="sin1reg Courbe et histogramme pour x1",data=sin1reg)
```

4 Package graphics R

4.1 Introduction

Le logiciel R est l'un des logiciels incontestablement reconnu pour ses graphiques tels que Les bandes linéaires , les bandes en pointillé ou même les bandes de nuages, ...



R a intégré des bibliothèques qui offrent un excellent support graphique. L'installation R contient trois packages importants :

1. les graphiques
2. les treillis
3. la grille

Ces packages fournissent des outils pour dessiner une grande variété de tracés et de formes. De plus, de nombreux packages externes tels que ggplot2 peuvent être téléchargés dans R.

QU'EST-CE QUE LE BUT D'UN GRAPHIQUE

- Mieux comprendre le problème
- Mieux expliquer un phénomène
- Le but est de nous nous aider à rendre une situation abstraite en information convaincante.

Pour faire ces graphiques , le logiciels "R" a mis à disposition un package à cet effet qui , de par ces multiples commandes met en exécution une fonction de tracé.
Nous allons donner une brève description de ces trois packages en mettant un accent particulier sur la commande "$PLOT$ " ci-dessous.

4.2 Description et installation d'un package graphics R

La bibliothèque graphique est le package graphique de base standard fourni avec l'installation R. Cette bibliothèque a été développée à l'origine pour R. De nombreux tracés standards

tels que des tracés de base de points et de lignes, des histogrammes, des graphiques à barres, des camemberts, des bplots, etc. peuvent être dessinés avec cette bibliothèque.
À toutes fins pratiques, la bibliothèque graphique est suffisamment silencieuse pour créer des tracés et des graphiques d'excellente qualité que nous utilisons généralement pour l'analyse des données et des statistiques.

Ce package est fourni avec l'installation de R.

Le paquet de grille

Le système graphique de grille a ensuite été développé par Paul Murrel et ajouté à R. Il s'agit d'un système graphique de bas niveau qui permet de dessiner et d'organiser des formes géométriques de base telles que des polygones, des courbes, des images raster, etc. Le paquet de grille contient des fonctions pour accéder au canevas et permet la création de plusieurs régions appelées fenêtres sur une seule toile.
Ce package est fourni avec l'installation R. Nous devons charger la bibliothèque de grille dans R avant de l'utiliser. (Le package graphique se charge automatiquement lorsque nous démarrons R).

Pour charger la bibliothèque de grille dans R, tapez

```
bibliothèque ("grille")
```

Le paquet lattice

Le treillis , développé par Deepayan Sarkar, est un système de visualisation de haut niveau basé sur une méthode appelée graphique en treillis . Ce package gère très efficacement les données multivariées.
Le package lattice se compose de fonctions de haut niveau pour chaque tâche. Ces fonctions renvoient des objets qui peuvent être convertis en graphiques par les fonctions `plot()` du package R de base. Ce package est basé sur le moteur graphique de grille mentionné ci-dessus.
Ce package est également fourni avec l'installation de R et nécessite le chargement du package `grDevices` .

Pour charger la bibliothèque grDevices dans R, tapez

```
bibliothèque ("grDevices")
```

Le paquet ggplot2

Le ggplot2 est une bibliothèque graphique pour R, créée par Hadley Wiskham. Il est mentionné dans sa page d'accueil que " ggplot2 est un système de traçage pour R, basé sur la grammaire des graphiques, qui essaie de prendre les bonnes parties des graphiques de base et de treillis et aucune des mauvaises parties". (Le Grammer of Graphics mentionné ici est un livre classique sur les méthodes graphiques pour la visualisation de données scientifiques écrit par Leyland Wilkinson).

Nous pouvons créer des parcelles très élégantes avec cette bibliothèque.

Le ggplot2 est un package externe qui doit être téléchargé de l'intérieur R. Pour installer ce package en ligne à partir de l'invite R, tapez

```
install.packages ("ggplot2")
```

4.3 Applications

R avec plot() , points() , lines() , polygon

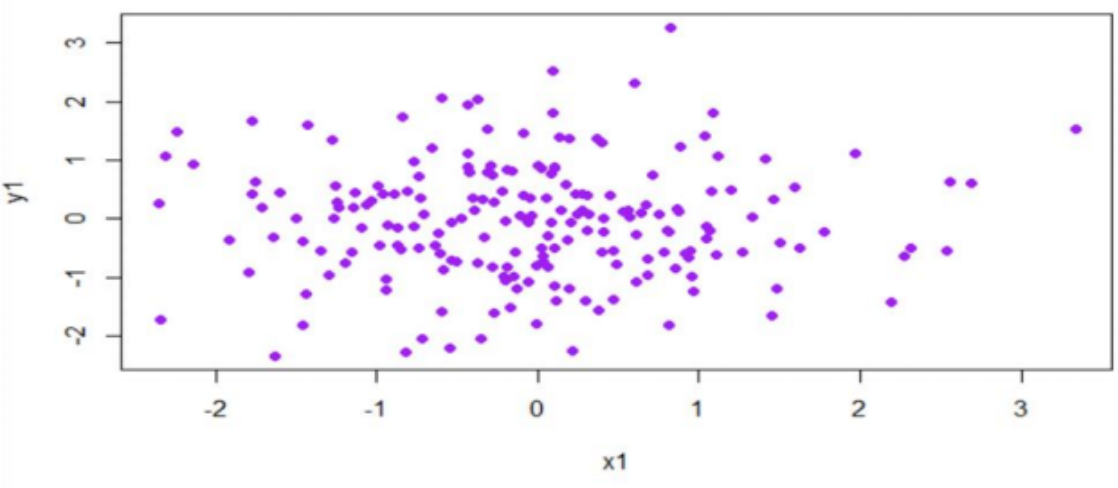
La fonction plot() trace les points et les lignes.

Les tracés de points et de lignes peuvent être produits en utilisant la fonction *terrain()*, qui prend les points x et y sous forme de vecteurs ou de nombre unique avec de nombreux autres paramètres. Les paramètres x et y sont nécessaires. Pour d'autres, la valeur par défaut sera utilisée en l'absence de la valeur. Dans les lignes de commande ci-dessous, nous créons d'abord une paire de séquences x et y et les passons en paramètres à la fonction *terrain()*.

Commande d'exécution de plot avec un graphique en nuage de point

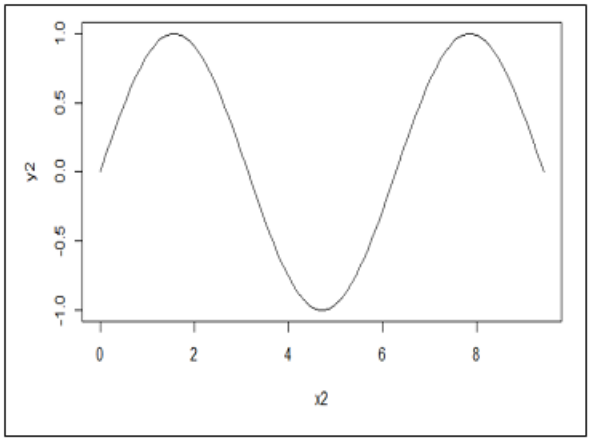
```
#Comment représenter graphiquement la commande plot avec des nuages de points
x1<-rnorm(200,mean = 0, sd=1)
y1<-rnorm(200,0,1)
plot(x1,y1,pch=16,col="purple")
```

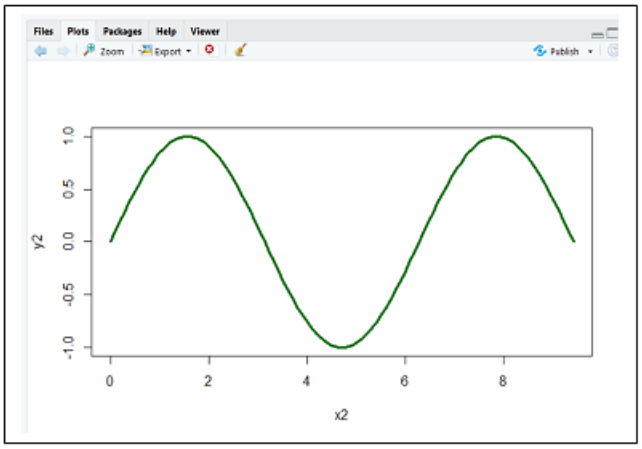
Résultat de l'application avec plot (Graphique nuage de point)



Commande d'exécution de plot avec un graphique en ligne

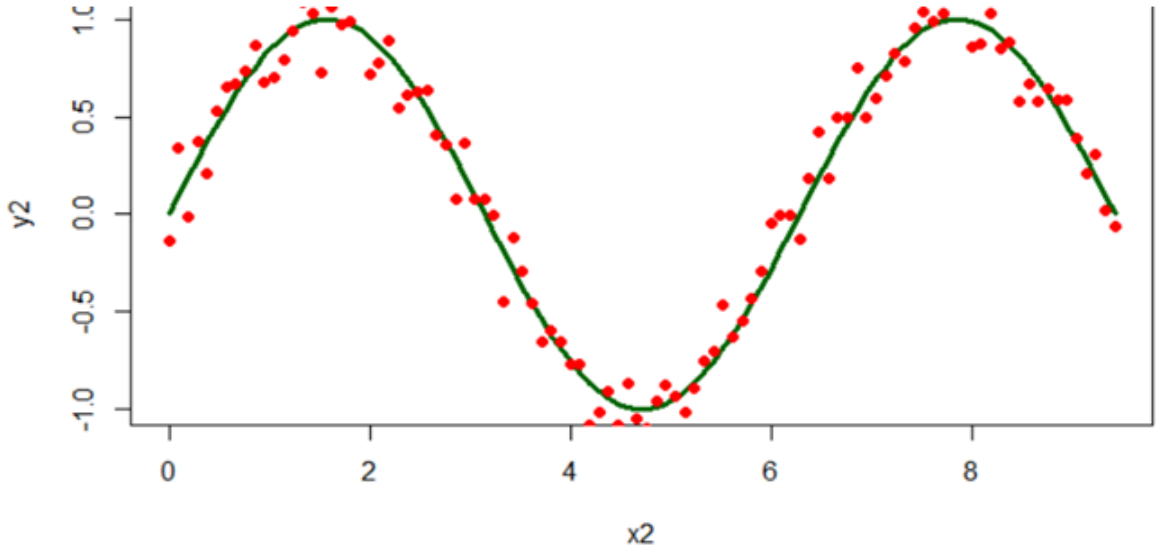
```
x2<-seq(0,3*pi,len=100)
y2<-sin(x2)
plot(x2,y2,type="l")
plot(x2,y2,type="l",lwd=3,col="darkgreen")
```





Exemple de graphique en pointillé pris aléatoirement

```
y2.rand<-y2+rnorm(100,0,0.1)
points(x2,y2.rand,pch=16,col="red")
```



♣ Bibliographie ♣

[1] Site le CRAN *https ://cran.r-project.org/web/packages/available packages by name.html available-packages-E*

[2] *https ://cran.r-project.org/web/packages/*

[3] Sidney I. Resnick *Extreme Values, Regular Variation and Point Processes*

[4] Stuart Coles *An Introduction to Statistical Modeling of Extreme Values*

[5] Laurens de Haan *Extreme value theory*

[6] *Documentation R sur Rminer*

[7] Cortez, P. *(2010). Data Mining with Neural Networks and Support Vector Machines using the R/rminer Tool. In Perner, P., editor, Advances in Data Mining Applications and Theoretical Aspects, 10th Industrial Conference on Data Mining, pages 572583, Berlin, Germany. LNAI 6171, Springer.*

[8] *Wikipedia*